

# Codd's Rules

## Rule 1: The information Rule

*"All information in a relational database is represented explicitly at the logical level in exactly one way – by values in tables"*

This rule specifies that 2-D tables (relations) are the only data structures used in a relational database. A relation is the definition of a table with attributes (columns) and tuples (rows) in it. This SQL query proves that the information in the mulcahyDentalPractice database conforms to this rule.

The SQL query "SHOW TABLES FROM mulcahyDentalPractice" displays the tables in the database.

```
>SHOW TABLES FROM mulcahyDentalPractice;
```

Tables_in_mulcahydentalpractice
appointments
bill
highrepayment
latecancel
latecancellations
laterepayment
nextweek_appointments
patient
payment
referrals
specialist
treatment

To prove this rule, I typed the query "SELECT \* FROM `table`". This displays all the information from the selected table.

```
>SELECT * FROM appointments;
```

appointmentNumber	appointmentDate appointment date	appointmentTime appointment time	patientNumber Patient Identifier	appointmentCancelDate enter cancellation date if cancelled, otherwise NU...	reminded	lateCancellation If late cancellation enter YES, if not type NO
1	2021-11-05	10:00:00	2004	NULL	0	NULL
2	2021-11-05	12:30:00	2006	NULL	0	NULL
3	2021-11-07	11:00:00	2001	2021-11-06	0	YES
4	2021-11-07	14:00:00	2010	NULL	0	NULL
5	2021-11-16	15:45:00	2001	2021-11-16	0	YES
6	2021-11-21	09:30:00	2001	2021-11-20	0	YES
7	2021-11-21	12:45:00	2002	NULL	0	NULL
8	2021-12-03	11:00:00	2017	NULL	0	NULL
9	2021-12-03	16:00:00	2012	NULL	0	NULL
10	2021-12-08	10:30:00	2015	NULL	0	NULL
11	2021-12-08	15:00:00	2013	NULL	0	NULL
12	2022-01-26	14:00:00	2014	NULL	0	NULL
13	2022-02-02	11:00:00	2015	2022-02-01	0	YES
14	2022-03-07	09:00:00	2016	NULL	0	NULL
15	2022-03-16	11:45:00	2020	NULL	0	NULL

This can be repeated with all the tables in the database.

## Rule 2: The Guaranteed Access Rule

*“Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name”*

All data in the SQL database can be accessed directly, only three items are needed to locate a piece of data: the table name, the primary key of the row, and the column name. Every row in a table should have a unique primary key. To insert a primary key into a table you type the following SQL query:

```
“ALTER TABLE `treatment` ADD PRIMARY KEY (`treatmentName`);”
```

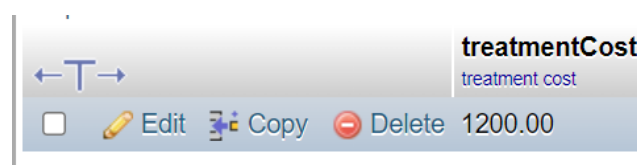
The primary key was already created in the database for all tables. There should be at least one primary key in each table of a database. It is important for each primary key to have a unique entry for each row in a table. If you try to insert a row with the same primary key as another row the action will not be carried out as you cannot put a primary key into the table twice.

Without a unique primary key, you can retrieve some data, but maybe not what you really wanted. It is not necessary that each row primary key should be unique, however, if it is not, the guaranteed access rule will not hold true. A DBS supports the concept of primary keys.

This rule can be proved by “logically accessing the database by resorting to a combination of table name, primary key value, and column name”. this rule was demonstrated simply by typing a SQL query to select one column from a table using table name, primary key, and column name. The SQL query typed was as follows:

```
“SELECT treatmentCost FROM treatment WHERE treatmentName='Dentures';”
```

where treatment is table name, treatmentCost is required field, and treatmentName is the primary key.



	treatmentCost
	1200.00

This rule can be demonstrated using any table name, primary , and column name in the database.

## Rule 3: Systematic Treatment of Null Values

*“Null values (distinct from empty character string or a string of blank characters or any other number) are supported in a fully relational DBMS for representing missing information in a systematic way, independent of data type.”*










A NULL value means the data for a particular field is unknown. The data may be unknown or inapplicable, however, it can't be filled in with 0 or a blank space. Primary keys cannot be NULL. To demonstrate this two rows were inserted into a table with blank space characters in a column in each table. A record was inserted into the treatment table but left the treatentCost blank and this displays as '0' in the database. The SQL query was as follows:

```
“INSERT INTO treatment (treatmentName, treatmentCost)
VALUES ('consultation', ' ');”
```

```
INSERT INTO treatment (treatmentName, treatmentCost) VALUES ('consultation', ' ');
```

[Edit inline](#) | [Edit](#) | [Create PHP code](#)

Warning: #1366 Incorrect decimal value: '' for column `mulcahydentelpractice`.`treatment`.`treatmentCost` at row 1


<input type="checkbox"/>	 <a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	Cleaning	50.00
<input type="checkbox"/>	 <a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	consultation	0.00
<input type="checkbox"/>	 <a href="#">Edit</a>	 <a href="#">Copy</a>	 <a href="#">Delete</a>	Crown	1000.00

Instead of the results coming up as '0' or '' it is important for it to come as NULL if there is no value for it (only should appear as 0 if the cost is 0). To add NULL values to a column you have to alter the table, so it accepts NULL values. If you add a NULL value to a record where it has not been altered to accept NULL values phpMyAdmin will not accept the NULL value in the query. To demonstrate this the following SQL query was typed:

```
“INSERT INTO treatment (treatmentName, treatmentCost)
VALUES ('Followup', NULL);”
```

But when this was typed this error came up:

```
INSERT INTO treatment (treatmentName, treatmentCost)
VALUES ('Followup', NULL);
```

**MySQL said:** 

#1048 - Column 'treatmentCost' cannot be null

To alter the table so that a column data type can be changed to accept a NULL value the following was typed into the console:

```
“ALTER TABLE treatment ALTER COLUMN 'treatmentCost' decimal (10,2) NULL;”
```

The row can now be updated, and a NULL value can be added to the blank fields.

A new record can be inserted into the database with NULL values in it. This is demonstrated by typing the following SQL query:

```
“INSERT INTO treatment (treatmentName, treatmentCost)
VALUES ('Followup', NULL);”
```

## Rule 4: Dynamic Online Catalogue based on the relational model

*“The data base description is represented at the logical level in the same way as ordinary data, so that authorised users can apply the same relational language to its interrogation as they apply to regular data.”*

We must be able to describe the structure of the database by looking at the metadata (data about the data). The metadata or structure of my database can be viewed by viewing the data dictionary.

The structure or metadata from a table can be accessed by typing the following SQL query:  
“SELECT \* FROM INNODB\_SYS\_TABLES WHERE TABLE\_ID = ‘tableID’

The metadata of a table within a database can be altered by using the ALTER TABLE clause on a table to change what data has been entered in that table.

## Rule 5: The Comprehensive Data Sub Language Rule

*“A relational system may support several languages and various modes of terminal use (for example fill-in-the-banks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is a comprehensive in supporting all of the following items: data definition, view definition, data manipulation (interactive and by program), integrity constraints, transaction boundaries (begin, commit, and rollback).”*

This defines the requirement for a language which can maintain database structural elements, modify, and retrieve data. The main language that is used in this database is SQL and this rule has been demonstrated through this language. I have also used PHP to demonstrate this rule on the website which is linked in the database via PHP. The database can be updated from the website via the PHP insert statement and the information can be selected from the database to be displayed via the PHP select statement.

Here is an example of a PHP code used to insert data into the database:

```
“mysql_select_db(“mulcahyDentalPractice”, $con);  
$sql = “INSERT INTO treatment (treatmentName, treatmentCost)  
VALUES  
('$ _POST[treatmentName]', '$ _POST[treatmentCost]’);”
```

This rule has been demonstrated by using PHP to carryout data manipulation such as creating a new record and updating them but also can be used in data retrieval transactions by the client. The view definition is being used as clients can use the PHP select statement to view certain data from the database.

SQL is the main type of language used to carry out tasks in the database. The comprehensive data sublanguage rule can be proven to work by typin in the following SQL queries into the console:

Being able to create a table with SQL is one way of demonstrating this rule:

```
“CREATE TABLE `bill` (  
  `billNumber` int(11) NOT NULL,  
  `billDate` date DEFAULT current_timestamp(),
```

```
`patientNumber` int(11) NOT NULL COMMENT,  
`appointmentNumber` int(11) DEFAULT NULL COMMENT,  
`treatmentName` varchar(255) DEFAULT NULL COMMENT); “
```

Then alter the table by adding a primary key:

```
“ALTER TABLE ‘bill’ ADD PRIMARY KEY(billNumber);”
```

Data is then inserted into the table:

```
INSERT INTO `bill` (`billNumber`, `billDate`, `patientNumber`, `appointmentNumber`,  
`treatmentName`) VALUES  
(1301, '2021-11-04', 2004, 1, 'Crown');”
```

## Rule 6: The View Updating Rule

*“All views that are theoretically updateable are also updateable by the system.”*

A view is a table that has been created by a SQL query. The view is like a virtual table which can take elements from multiple columns from one or more tables and display them in a separate view table. When the view table is created it is stored in the metadata in the database. A view table can also be update, and the DBMS should be able to handle the update, and if it is update properly then the base table should be able to update also.

It was first demonstrated that a view table should be created in the database by typing in the following SQL code into the console:

```
“CREATE VIEW `nextweek_appointments` (  
`patientNumber` int(11)  
, `appointmentDate` date  
, `appointmentTime` time  
, `reminded` tinyint(1)  
);”
```

A SELECT statemen can be carried out in the view:

```
“SELECT * FROM ‘nexterrk_appointments’ WHERE patientNumber = 2001;”
```

## Rule 7: High Level Insert Update and Delete Rule

*“The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.”*

High level means that you can affect multiple rows from multiple tables with a single query. To prove this rule the user must be able to multi-insert data, multi-update data, and multi-delete data in multiple tables and rows in the database rather than just for a single row in a single table.

To prove this, I typed the following SQL query into the console:

```
INSERT INTO patient VALUES (2021, 2022), (Martin, Mary), (Finnegan, Finnegan), (1984-  
04-21, 1986-07-05), (876562082, 868835852), (H22XE35, H22XE35),  
(m.finnegan6@gmail.com, m.finnegan6@gmail.com), (0.00, 0.00), (0, 0);
```

## Rule 9: Logical Data Independence

*“Applications programs and terminal activities remain logically unimpaired when Information-Preserving changes of any kind that theoretically permit unimpairment are made to the base tables.”*

If a change is made to the logical level (tables, rows, structure) or the base tables, such as adding a new column to one of the tables then data should still be able to be added to the table even though the structure has been changed.

I demonstrated this by adding a column to the patient table using the following SQL command:

```
ALTER TABLE patient ADD gender VARCHAR(10) DEFAULT NULL;
```

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0397 seconds.)

```
ALTER TABLE patient ADD gender VARCHAR(10) DEFAULT NULL;
```

	patientNumber	patientFirstName <small>Forename</small>	patientLastName <small>Surname</small>	patientDOB <small>DOB</small>	patientPhone <small>phone number</small>	patientEircode <small>eircode</small>	patientEmail <small>email address</small>	patientBalance <small>Amount patient owes</small>	if 1 - patient owes too much or for too long	gender
Edit Copy Delete	2001	Gary	Keaveney	1943-07-12	857496234	H23DY90	gkeave43@gmail.com	30.00	0	NULL
Edit Copy Delete	2002	Betty	Cummins	2002-09-05	838656517	H13DL91	bettyxcummins12x@gmail.com	0.00	0	NULL
Edit Copy Delete	2003	Darragh	Leonard	2005-05-12	874228692	H34ED95	lendarragh@gmail.com	50.00	0	NULL
Edit Copy Delete	2004	George	Leonard	1977-10-14	874228692	H34ED95	gleonard77@gmail.com	100.00	0	NULL
Edit Copy Delete	2005	Meadhbh	Leonard	1980-05-15	874228692	H34ED95	moconnell22@gmail.com	0.00	0	NULL
Edit Copy Delete	2006	Ciara	Leonard	2011-02-19	874228692	H34ED95	moconnell22@gmail.com	0.00	0	NULL
Edit Copy Delete	2007	Karen	Carroll	1951-11-30	868932058	H12FX01	karcarr@yahoo.com	0.00	0	NULL
Edit Copy Delete	2008	Becky	McLoughlin	1996-03-06	858937878	H18YLW5	bmcloughlin@gmail.com	0.00	0	NULL
Edit Copy Delete	2009	Milo	Costelloe	1994-04-28	873357130	H13DD00	m.costelloe7@ucc.ie	0.00	0	NULL

I should now be able to insert a new row into the tenant table with no problems occurring.  
 INSERT INTO patient (patientNumber, patientFirstName, patientLastName, patientDOB, patientPhone, patientEircode, patientEmail, patientBalance, defaulter, gender)  
 VALUES(2021, 'James', 'Joyce', '1999-10-12', 876462974, 'H23GY45', 'jjoyce@gmail.com', '0.00', 0, 'Male');

✓ 1 row inserted. (Query took 0.0093 seconds.)

```
INSERT INTO patient (patientNumber, patientFirstName, patientLastName, patientDOB, patientPhone, patientEircode, patientEmail, patientBalance, defaulter, gender) VALUES(2021, 'James', 'Joyce', '1999-10-12', 876462974, 'H23GY45', 'jjoyce@gmail.com', '0.00', 0, 'Male');
```

Edit Copy Delete	2019	Andy	Muldoon	1988-08-12	856532588	H15MUJ34	andrewmuldoonoon@live.com	0.00	0	NULL
Edit Copy Delete	2020	Joe	Burke	1949-01-01	862340983	H15UQ14	NULL	75.00	0	NULL
Edit Copy Delete	2021	James	Joyce	1999-10-12	876462974	H23GY45	jjoyce@gmail.com	0.00	0	Male

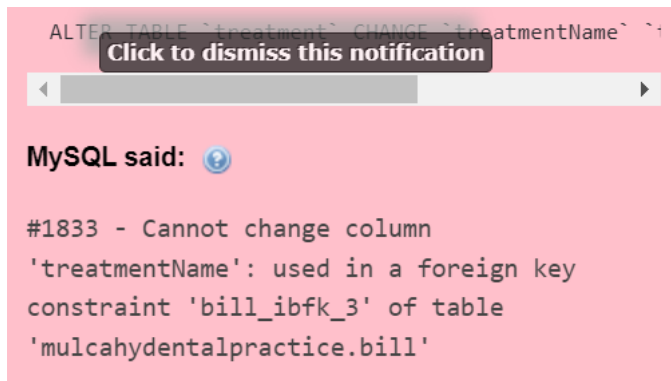
## Rule 10: Integrity Independence

*"Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalogue, not in the application programs."*

A primary key cannot contain a NULL value, to prove this I attempted to change a primary key to a NULL value:

```
ALTER TABLE treatment ALTER COLUMN treatmentName int(30) NULL;
```

I encountered an error and was unable to do this



But had this not been a foreign key in a different table I would have been able to update this to accept NULL values. I then would have entered the following query seen the following error:

```
INSERT INTO treatment (treatmentName, treatmentCost)
VALUES (NULL, '30.00');
"#1048 – Column `treatmentName` cannot be null"
```

Primary keys cannot have NULL values.

### Rule 11: Distributed Independence:

*"A relational DBMS has distribution independence."*

The DBMS should appear like one database and users and applications are not required to know where only the data is stored. The end user of the DBMS must not be able to see the data is distributed over various locations. The user should always get the impression that the data is located at one site.

### Rule 12: Non Subversion Rule

*"If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity or constraints expressed in the higher-level relational language (multiple-records-at-a-time)."*

There is no way to modify this database's structure other than using SQL.